UDC: 004.4’24, 004.896 https://doi.org/10.15407/jai2023.02.066

ІНТЕГРАЦІЯ МЕТОДІВ АЛГЕБРИ АЛГОРИТМІВ ТА ОБЧИСЛЮВАЛЬНОГО ІНТЕЛЕКТУ ДЛЯ АВТОМАТИЗАЦІЇ ПРОЄКТУВАННЯ ПРОГРАМНИХ СИСТЕМ

А.Ю. Дорошенко[1], І.П. Сініцин[2], В.Л. Шевченко[3], О.А. Яценко[4], І.З. Ашур[5]
[1,2,3,4] Інститут програмних систем НАН України, Україна
пр. Академіка Глушкова, 40, м. Київ, 03187
[1,2,5] Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського», Україна
пр. Перемоги, 37, м. Київ, 03056
[1]a-y-doroshenko@ukr.net
[2]ips2014@ukr.net
[4]oayat@ukr.net,
[5]ilyaachour@gmail.com
[1]https://orcid.org/0000-0002-8435-1451
[2]https://orcid.org/0000-0002-4120-0784
[3]https://orcid.org/0000-0002-9457-7454
[4]https://orcid.org/0000-0002-4700-6704
[5]https://orcid.org/0000-0003-2348-8777

Анотація. Розглянуто підхід до створення інтелектуальних систем, що поєднує ідеї алгебри алгоритмів Глушкова з відомими нейроеволюційними алгоритмами обчислювального інтелекту, що можуть застосовуватись для автоматизації проєктування та синтезу програм. Метод нейроеволюції наростаючих топологій призначений для зменшення розмірності простору пошуку параметрів нейромережі у вигляді поступового розвитку її структури у процесі еволюції. Розроблено програмний інструментарій на підтримку такого підходу, де в основу покладено конструювання високорівневих специфікацій алгоритмів, представлених у системах алгоритмічних алгебр, та генерацію відповідних програм на основі шаблонів реалізацій цільовою мовою програмування. Використовуються засоби гіперсхем — параметризованих алгоритмів для розв’язання певного класу задач. Установка конкретних значень параметрів і наступна інтерпретація гіперсхем дозволяє одержати алгоритми, адаптовані до конкретних умов застосування. Розроблений інструментарій проєктування та синтезу програм забезпечує покрокову розробку програм, починаючи від високорівневої специфікації, поданої алгебро-алгоритмічною мовою. На виході інструментарію автоматизовано генерується програма однією з цільових мов програмування (C, C++, Java, Python та ін.), до яких належить також предметно-орієнтована мова проєктування нейромереж. Згадана мова включає оператори та умови роботи з популяціями, конфігураціями, геномами та функцією приданості. Роботу інструментарію проілюстровано на прикладі проєктування параметризованого алгоритму оцінювання для двійкового мультиплексора та генерації програми для задачі балансування зі зворотним маятником.

Ключові слова: алгебра алгоритмів, генерація програм, нейроеволюця, нейромережа, штучний інтелект.

INTEGRATION OF THE METHODS OF ALGEBRA OF ALGORITHMS AND COMPUTATIONAL INTELLIGENCE FOR AUTOMATION OF PROGRAM SYSTEMS DESIGN

A.Yu. Doroshenko [1], I.P. Sinitsyn [2], V.L. Shevchenko [3], O.A. Yatsenko [4], I.Z. Achour[5]
[1,2,3,4] Institute of Software Systems of National Academy of Sciences of Ukraine, Ukraine
Glushkov ave., 40, Kyiv, 03187
[1,2,5] National Technical University of Ukraine «Igor Sikorsky Kyiv Polytechnic Institute», Ukraine
Peremohy ave., 37, Kyiv, 03056
[1]a-y-doroshenko@ukr.net
[2]ips2014@ukr.net
[4]oayat@ukr.net
[5]ilyaachour@gmail.com
[1]https://orcid.org/0000-0002-8435-1451
[2]https://orcid.org/0000-0002-4120-0784
[3]https://orcid.org/0000-0002-9457-7454

[4]https://orcid.org/0000-0002-4700-6704
[5]https://orcid.org/0000-0003-2348-8777

Abstract. An approach to the creation of intelligent systems is considered, which combines the ideas of Glushkov's algebra of algorithms with the well-known neuroevolutionary algorithms of computational intelligence, which can be used to automate the design and synthesis of programs. The method of neuroevolution of augmenting topologies is intended to reduce the dimensionality of the space for searching for neural network parameters in the form of gradual development of its structure in the process of evolution. A software toolkit has been developed to support the approach, which is based on the construction of high-level specifications of algorithms represented in systems of algorithmic algebras, and generation of corresponding programs based on implementation templates in a target programming language. Parameterized algorithms called hyperschemes are used to solve a certain class of problems. Setting specific values of parameters and subsequent interpretation of hyperschemes allows obtaining algorithms adapted to specific application conditions. The developed program design and synthesis toolkit provides step-by-step development of programs, starting from a high-level algebraic-algorithmic specification. At the output of the toolkit, a program is automatically generated in one of the target programming languages (C, C++, Java, Python), that also include a subject-oriented language for designing neural networks. The mentioned language includes operators and conditions for working with populations, configurations, genomes, and fitness function. The work of the toolkit is illustrated by the example of designing a parameterized evaluation algorithm for a binary multiplexer and generation of a program for the single-pole balancing problem.

Keywords: algebra of algorithms, program generation, neuroevolution, neural network, artificial intelligence.

Вступ

В процесі розвитку робіт в області штучного інтелекту на початку 90-х рр. шляхом інтеграції низки інтелектуальних технологій та методів сформувався новий напрямок, що отримав назву *обчислювального інтелекту*. Обчислювальний інтелект [1, 2] є сукупністю технологій, моделей, методів і програмних засобів, призначених для розв'язання неформальних, творчих задач у різних сферах людської діяльності з використанням засобів нечіткої логіки, штучних нейронних мереж, еволюційних обчислень, машинного навчання, ймовірнісних методів, що певною мірою моделюють види розумової діяльності людини (такі як нечіткість міркувань, якісний та інтуїтивний підходи, креативність, логічний вивід, самонавчання і т. ін.), зокрема, в галузі прийняття рішень, класифікації, розпізнавання образів і т. д.

У даній роботі ми зосереджуємось навколо одного з центральних понять обчислювального інтелекту, а саме — поняття штучної нейронної мережі, що означає граф вузлів, об'єднаних зв'язками, кожен з яких має певну вагу [3]. Вузол нейромережі є свого роду пороговим оператором, який дозволяє сигналу проходити далі тільки після спрацьовування певної функції активації. Як правило, процес навчання нейромережі складається з вибору відповідних значень ваги для всіх зв'язків у мережі. Таким чином, нейромережа здатна апроксимувати будь-яку функцію і може розглядатися як універсальний апроксиматор, який визначається теоремою універсальної апроксимації.

За останні кілька десятків років було винайдено багато методів навчання нейромереж. Одна з найпопулярніших технік, що здобула поширення за останні десятиліття, була запропонована Д. Хінтоном [4]. Вона ґрунтується на зворотному поширенні помилки прогнозування через мережу з різними методами оптимізації, побудованими на основі градієнтного спуску функції втрат по відношенню до ваг зв'язків між вузлами мережі. Поряд з методами зворотного поширення застосовуються дуже багатообіцяючі еволюційні алгоритми [3], які можуть вирішувати вищезгадані проблеми. Ці методи використовують принципи еволюції видів для розробки штучних нейронних мереж.

Основна ідея *нейроеволюції* полягає в тому, щоб створювати нейромережі за допомогою стохастичних методів пошуку, що ґрунтуються на популяції. Використовуючи еволюційний підхід, можна розробити оптимальні архітектури нейронних мереж, які точно вирішують конкретні завдання. В результаті можуть бути створені компактні та

енергоефективні мережі з помірними вимогами до обчислювальної потужності. Процес еволюції реалізується шляхом застосування генетичних операторів (мутація, кросовер) до популяції хромосом (генетично кодовані уявлення нейромереж або рішень) протягом багатьох поколінь. Щоб усунути недоліки методів з фіксованою топологією, Кеннет О. Стенлі запропонував особливий метод нейроеволюції з розвитком топології NeuroEvolution of Augmenting Topologies, NEAT) [5, 6]. Основна ідея цього алгоритму у тому, що еволюційні оператори застосовуються як до вектору з вагами всіх зв'язків, так і до топології створеної нейронної мережі.

Дана робота присвячена розробці засобів автоматизації проєктування програм, що використовують нейроеволюційні алгоритми. Одним з напрямів у розробці та дослідженні програмних систем є побудова програмних абстракцій у вигляді алгебро-алгоритмічних мов і моделей, що ставить своєю метою розвиток архітектурно- і мовно-незалежних засобів програмування. У попередніх роботах авторів [7, 8, 9] були запропоновані теорія, методологія та інструментарій для автоматизованого проєктування паралельних програм, що ґрунтуються на засобах високорівневої алгебро-алгоритмічної формалізації. Зокрема, в Інституті програмних систем НАНУ було розроблено інструментальну систему автоматизації програмування, названу інтегрованим інструментарієм проєктування та синтезу програм (англ. Integrated Toolkit for Designing and Synthesis of programs, IDS Toolkit). Інструментарій забезпечує покрокову розробку програм, починаючи від високорівневої алгебро-алгоритмічної специфікації і закінчуючи кодом цільовою мовою програмування (зокрема, C++, Java, Python). У даній роботі розглянуті результати застосування інструментарію для автоматизованої розробки нейроеволюційних програм, що використовують бібліотеки SharpNEAT [10] та NEAT-Python [11], які є реалізаціями алгоритму NEAT мовами програмування C# та Python, відповідно.

1. Алгебро-алгоритмічний інструментарій автоматизації проєктування програм

Математичним базисом для формалізації та автоматизації проєктування алгоритмів та програм в даній роботі є системи алгоритмічних алгебр (САА) Глушкова [7, 8], що виступають універсальним засобом високорівневих специфікацій (САА-схем) будь-яких алгоритмів, включно з послідовними й паралельними імперативними, а також нейромережевими алгоритмами. Згадані системи покладені в основу алгоритмічної мови САА/1, призначеної для багаторівневого структурного проєктування й документування послідовних і паралельних алгоритмів та програм.

Основними операторними конструкціями мови САА/1, що застосовуються в даній роботі, є такі:

– композиція (послідовне виконання) операторів: “*operator* 1”; “*operator* 2”;

– умовний оператор: IF ‘*condition*’ THEN “*operator* 1” ELSE “*operator* 2”;

– цикл: FOR EACH (“*variable* 1” IN “*expression* 1”) LOOP “*operator* 1” END OF LOOP.

Перевагою використання мови САА/1 є можливість подання ідентифікаторів операторів та умов у формі, близькій до природної мови. В подвійних лапках в схемах алгоритмів вказуються ідентифікатори операторів, в одинарних — умови (предикати).

Однією з важливих проблем в рамках алгебро-алгоритмічного підходу є підвищення адаптивності програм до конкретних умов їхнього використання. Зокрема, вона вирішується за рахунок застосування параметрично-керованої генерації специфікацій алгоритмів на основі схем більш високого рівня, що називаються гіперсхемами [7, 8]. *Гіперсхеми* є параметризованими алгоритмами для розв'язання певного класу задач. Установка конкретних значень параметрів і наступна

інтерпретація гіперсхем дозволяє одержати алгоритми, адаптовані до конкретних умов застосування.

Розроблений інструментарій проєктування та синтезу програм IDS забезпечує покрокову розробку програм, починаючи від високорівневої алгебро-алгоритмічної специфікації, поданої мовою САА/1. На виході інструментарію автоматизовано генерується програма однією з цільових мов програмування (C, C++, Java, Python та ін.), до яких належить також предметно-орієнтована мова проєктування нейромереж. Згадана мова включає оператори та умови роботи з популяціями, конфігураціями, геномами та функцією приданості (fitness), зокрема:

– "Create feedforward neural network (*net*) for (*genome*) and configuration (*config*)";
– "Create the population (*p*), which is the top-level object for a NEAT run, for configuration (*config*)";
– 'Genome (*fitness*) is greater or equal to fitness threshold (*fitness_threshold*) for configuration (*config*)';
– 'Genome (*fitness*) is less than fitness threshold (*fitness_threshold*) for configuration (*config*)'.

Процес автоматизованої розробки алгоритмів та програм в інструментарії зображено на рис. 1.

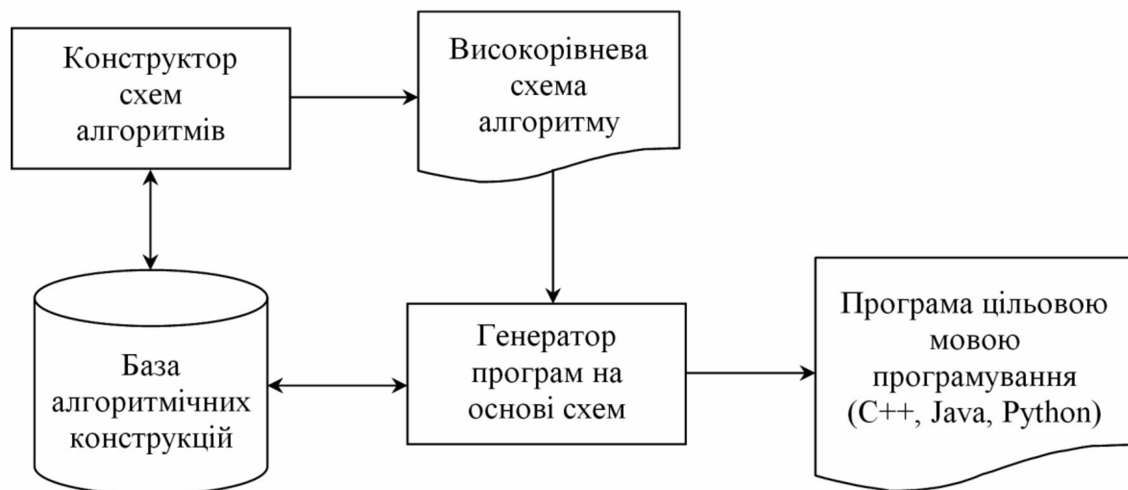


Рис. 1. Проєктування алгоритмів та синтез програм в системі IDS

В розділах 2 та 3 розглянуті приклади застосування інтегрованого інструментарію для розробки програм, що використовують нейроеволюційні алгоритми.

2. Застосування інструментарію для генерації алгоритмів оцінки двійкового мультиплексора

У даному розділі інструментарій алгебр алгоритмів використано для параметрично-керованої генерації САА-схем оцінювання для двійкового мультиплексора (BinaryMultiplexer-Evaluator) на основі гіперсхем з подальшою генерацією коду мовою програмування із використанням методів фреймворку SharpNEAT [10]. Даний фреймворк є реалізацією алгоритму нейроеволюції наростаючих топологій мовою C# для платформи NET, розробленою К. Гріном.

Мультиплексор — пристрій, що має декілька входів даних x_i $(i = 0,\ldots,n-1)$, адресні входи s_j $(j = 0,\ldots,m-1)$, та один вихід y. Пристрій передає сигнал з одного зі входів даних на вихід; при цьому вибір потрібного входу здійснюється шляхом подачі відповідної комбінації керуючих сигналів на адресних входах. Кількість входів даних n та кількість адресних входів m зв'язані співвідношенням $n = 2^m$.

Фрагмент побудованої за допомогою інтегрованого інструментарію гіперсхеми

для генерації алгоритмів оцінювання двійкового мультиплексора наведений далі. Параметрами гіперсхеми є такі: *P1* — кількість адресних входів мультиплексора; *P2* — кількість інформаційних входів; *P3* = *P1* + *P2* — загальна кількість входів. Всі входи приймають двійкові значення (0 або 1). Двійкова адреса подається на адресні входи, що репрезентує вибір одного зі значень входів для даних. Оцінка складається з вичерпної перевірки нейронної мережі на кожній з 2^{P3} можливих комбінацій входів. Вихідне значення нейронної мережі повинне збігатися зі значенням одного зі входів даних, що представлений двійковою адресою з адресних входів. Вихідне значення, менше за 0.5, вважають двійковим нулем, вихідне значення, більше або рівне 0.5, — двійковою одиницею. Значення оцінки придатності адитивно розраховують у результаті вичерпної перевірки.

```
SCHEME BINARY{P3}MULTIPLEXEREVALUATOR ====
"Binary{P3}MultiplexerEvaluator"
==== NAME SPACE SharpNeat.Domains.Binary{P3}Multiplexer (
CLASS Binary{P3}MultiplexerEvaluator OF TYPE public
INHERITS IPhenomeEvaluator<IBlackBox>
"Declare a constant (StopFitness) of type (double) = (10E+[P1])";
"Declare a variable (_evalCount) of type (ulong)";
"Declare a variable (_stopConditionSatisfied) of type (bool)";
REGION IPhenomeEvaluator<IBlackBox> Members
METHOD public FitnessInfo Evaluate(IBlackBox box)
"Declare a variable (fitness) of type (double) = (0.0)";
"Declare a variable (success) of type (bool) = (true)";
"Declare a variable (output) of type (double)";
"Declare a variable (inputArr) of type (ISignalArray) =
(box.InputSignalArray)";
"Declare a variable (outputArr) of type (ISignalArray) =
(box.OutputSignalArray)";
"Increase (_evalCount) by (1)";
FOR (i FROM 0 TO [Pow(2, P3)-1]) LOOP
"Declare a variable (tmp) of type (int) = (i)";
FOR (j FROM 0 TO [P3-1]) LOOP
(inputArr[j] := tmp&0x1); (tmp := tmp >> 1)
END OF LOOP;
"Activate the black box (box)";
"Read output signal (output)(outputArr)";
IF (((1<<([P1]+(i&0x[P2-1])))&i) != 0)
THEN
(fitness := fitness + 1.0 - ((1.0 - output) * (1.0 - output)));
IF (output < 0.5) THEN (success := false) END IF
ELSE
(fitness := fitness + 1.0 - (output * output));
IF (output >= 0.5) THEN (success := false) END IF
END IF;
"Reset black box state ready for next test case (box)"
END OF LOOP;
IF success THEN (fitness := fitness + 10E+[P1]) END IF;
IF (fitness >= StopFitness) THEN (_stopConditionSatisfied := true)
END IF;
"Return value (new FitnessInfo(fitness, fitness))"
END OF METHOD
END OF REGION
END OF CLASS )
END OF SCHEME BINARY{P3}MULTIPLEXEREVALUATOR
```

На основі гіперсхеми в інтегрованому інструментарії були згенеровані САА-схеми оцінки мультиплексорів з трьома, шістьма та одинадцятьма входами відповідно до встановлених значень параметрів *P1*, *P2*,

P3. За схемами був згенерований програмний код мовою C# для SharpNEAT [10].

Далі наведено результати експериментів з виконання згенерованої програми оцінки мультиплексора з одинадцятьма входами при використанні однопроцесорної багатопотокової реалізації нейроеволюції наростаючих топологій, розробленої К. Гріном [11], та розподіленої реалізації, запропонованої в [12]. У якості середовищ для виконання багатопотокової та розподіленої реалізації були обрані такі конфігурації:

– локальне середовище, Intel Core i9-9900K CPU (3.60 ГГц – 5.00 ГГц), 8 ядер, 16 логічних процесорів, 32.0 ГБ RAM. Один процес, багатопотокова реалізація, 16 потоків;

– те ж саме локальне середовище, розподілена реалізація, 16 локальних клієнтів-виконувачів;

– хмарне середовище, 3rd Gen AMD EPYC Amazon EC2 C6a.large, 3.60 ГГц, 2 ядра, 4.0 ГБ RAM, до 12.5 Гбіт/с мережевої пропускної здатності та до 6600 Мбіт/с пропускної здатності сховища. Розподілена реалізація з кількістю хмарних клієнтів-виконувачів 16, 32 та 64.

На рис. 2 зображено графік залежності швидкості оцінювання (кількості оцінювань в секунду) від номера покоління для локальних конфігурацій середовища. Як видно з рисунка, розподілена реалізація очікувано демонструє гірші результати у порівнянні з однопроцесорною реалізацією у зв'язку з наявністю накладних витрат на взаємодію між процесами. З ростом складності задачі оцінювання (ростом розміру згенерованої нейромережі) ефективність однопроцесорної та локальної розподіленої реалізації вирівнюється, оскільки накладні витрати обчислювальних ресурсів стають непомірно меншими за витрати на оцінювання.

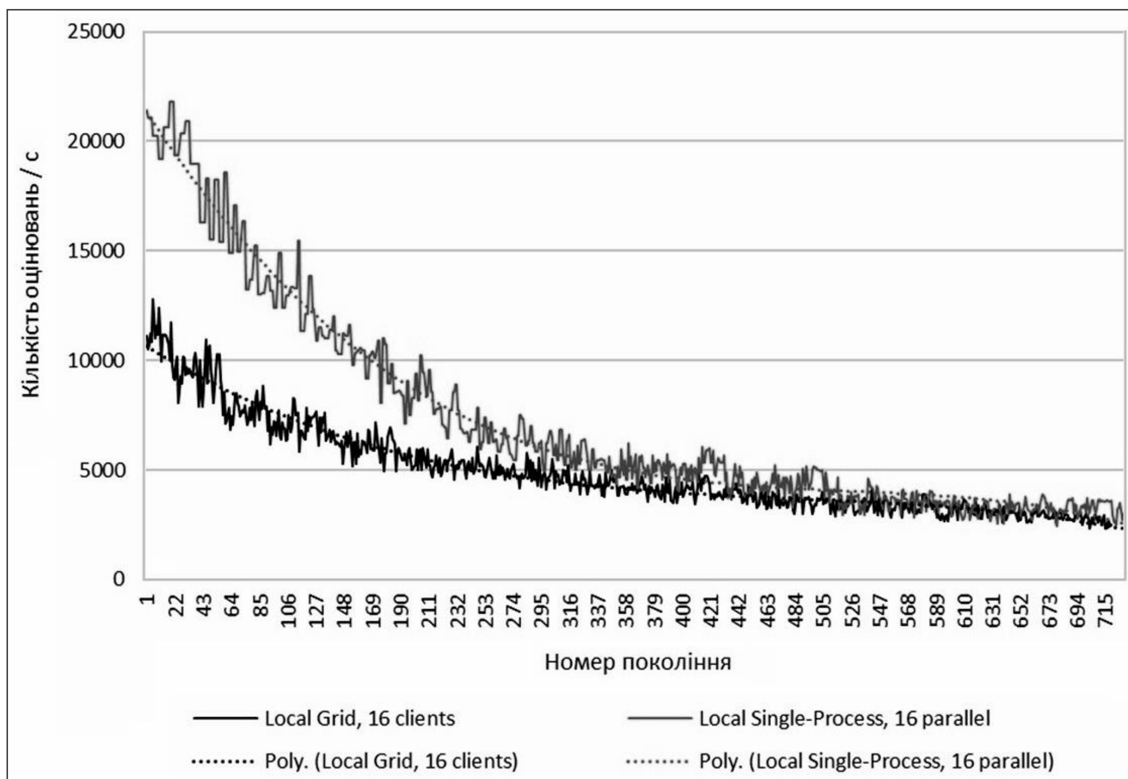


Рис. 2. Графік залежності швидкості оцінювання від номера покоління для локальних конфігурацій середовища

На рис. 3 зображено графік залежності швидкості оцінювання від номера покоління для хмарних конфігурацій середовища.

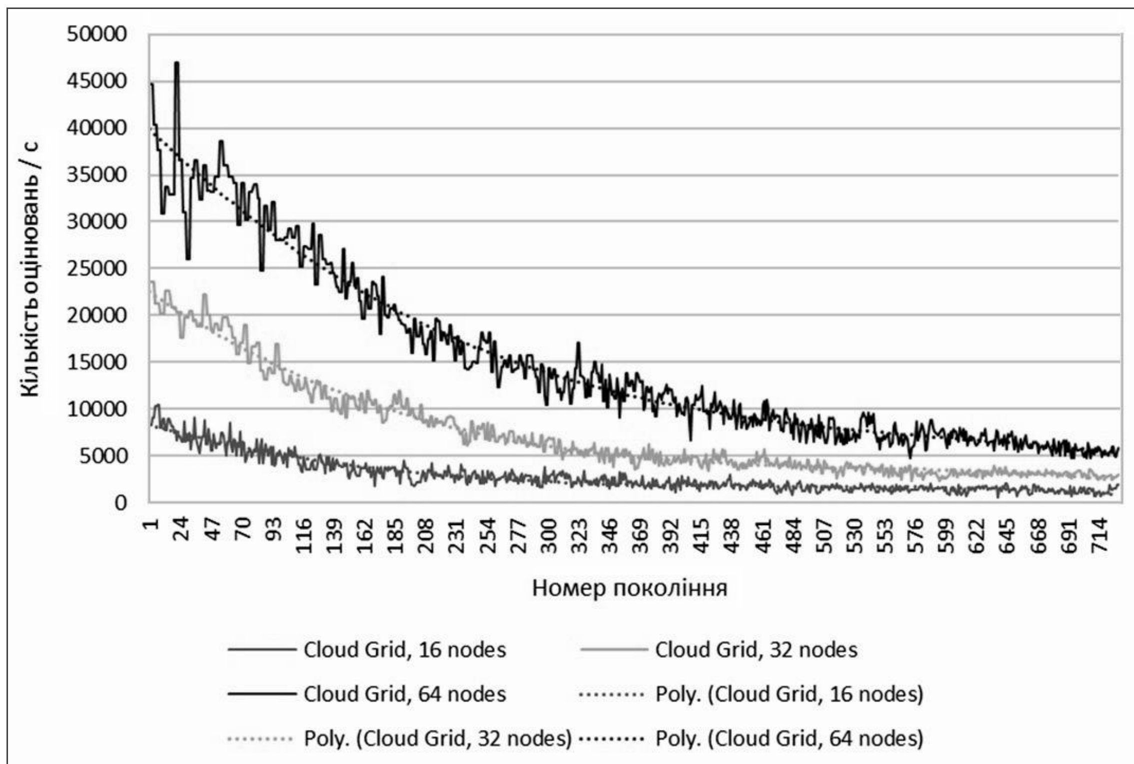


Рис. 3. Графік залежності швидкості оцінювання від номера покоління для хмарних конфігурацій середовища

Як видно з графіка, розподілена хмарна реалізація очікувано демонструє гірші результати (за тієї ж кількості клієнтів-виконувачів) у порівнянні з однопроцесорною та локальною розподіленою реалізацією у зв'язку з наявністю накладних витрат на взаємодію між процесорами багатьох клієнтів-виконувачів. Проте, з ростом кількості клієнтів-виконувачів ми можемо нехтувати сталим значенням накладних витрат і отримувати лінійний ріст ефективності розподіленої системи.

Даний експеримент продемонстрував можливість розподіленої системи проводити оцінювання на 64-ох хмарних клієнтах-виконувачах і отримувати приріст у 60–100 % від максимальних можливостей однопроцесорної локальної реалізації.

3. Застосування алгебро-алгоритмічного інструментарію для задачі балансування

У даному розділі розглядається використання інтегрованого інструментарію для прикладної задачі балансування візка зі зворотним маятником. У даному експерименті нейроеволюційний алгоритм застосовується для реалізації контролера, що керує візком. Постановка задачі балансування зворотного маятника та використовувана математична модель (рівняння руху) детально розглянуті в [3]. Контролер балансування маятника приймає масштабовані вхідні значення і видає вихідний сигнал, що є двійковим значенням і визначає дію, яка має бути застосована у певний момент часу.

Початкова конфігурація нейромережі контролера може бути представлена у вигляді схеми на рис. 4. Вона включає п'ять вхідних вузлів: для горизонтального положення візка (x1) і його швидкості (x2), для вертикального кута маятника (x3) і його кутової швидкості (x4) і додатковий вхідний вузол для зміщення (x0) (яке може бути необов'язковим залежно від конкретної використовуваної бібліотеки NEAT). Вихідний вузол (a) є двійковим вузлом, що видає керуючий сигнал (0 або 1). Прихований вузол (h) є необов'язковим і може бути пропущений.

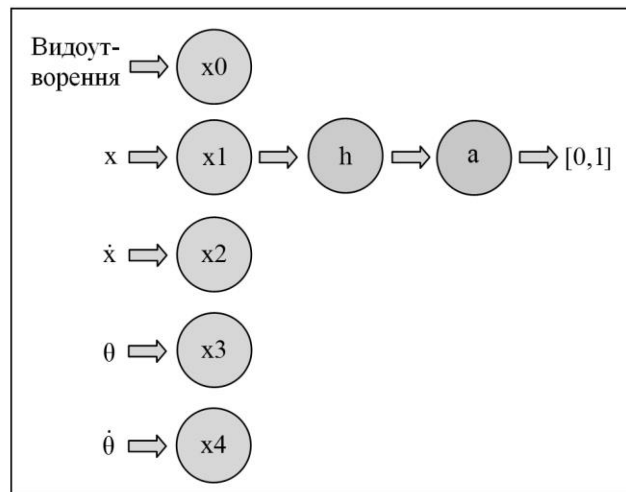


Рис. 4. Початкова конфігурація нейромережі балансувальника

Фрагмент однієї із САА-схем, що реалізує експеримент, наведений нижче. Функція (підпрограма) "Run experiment" починається із завантаження гіперпараметрів з файлу конфігурації і породжує початкову популяцію, використовуючи завантажену конфігурацію. Після цього функція налаштовує репо́ртери для збирання статистики, що стосується виконання еволюційного процесу. Процес еволюції виконується протягом вказаної кількості поколінь, а результати зберігаються у вихідному каталозі. Після того, як в ході еволюційного процесу знайдений найкращий геном, виконується перевірка, чи відповідає він критеріям порогу придатності, встановленому у файлі конфігурації. Програма виводить геном с формально найкращою відповідністю.

"Run experiment (*config_file*, *n_generations*=100)"
==== "Load configuration (*config*) from file (*config_file*)";
"Create the population (*p*), which is the top-level object for a NEAT run, for configuration (*config*)";
"Add a stdout reporter to show progress in the terminal for population (*p*)";
"Run neuroevolution for up to (*n*) generations and display the best genome (*best_genome*) among generations";
"Create feedforward neural network (*net*) for (*best_genome*) and configuration (*config*)";
"Output the message (\n\nEvaluating the best genome in random runs)";
(*success_runs* := "Evaluate best net (*net*, *config*, *additional_num_runs*)");
"Output successful (*success_runs*) and expected runs (*additional_num_runs*) and check if the best genome is a winner";
"Visualize the experiment results for configuration (*config*), best genome (*best_genome*) from directory (*out_dir*)";

На основі сконструйованих схем алгоритмів за допомогою інтегрованого інструментарію була виконана генерація програмного коду мовою Python. Далі наведено результати експерименту з виконання згенерованої програми. У файлі конфігурації NEAT-Python було визначено популяцію з 150 окремих організмів та встановлено поріг придатності зі значенням 1.0 в якості критерію завершення. Значення початкових параметрів нейромережі були такими: кількість прихованих вузлів — 0, вхідних — 4, вихідних — 1. Граф нейромережі контролера-переможця балансування одиночного зворотного маятника наведений на рис. 5.

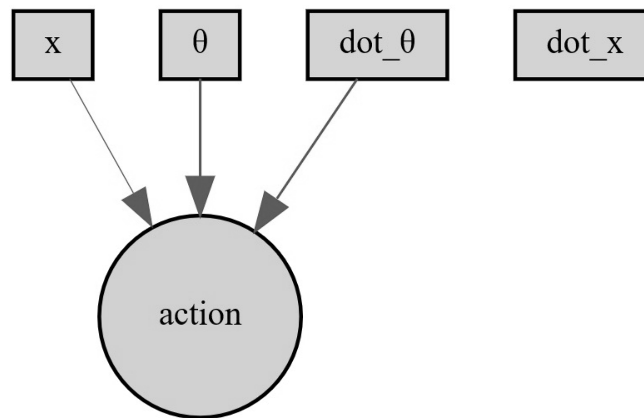


Рис. 5. Граф оптимального контролера балансування зворотного маятника, знайдений нейроеволюційним алгоритмом

Графік зміни значень придатності на протязі поколінь еволюції показано на рис. 6. Як видно з графіка, середня придатність популяції у всіх поколіннях була низькою, але з самого початку виникла корисна мутація, що породила певну лінію організмів. Із покоління в покоління обдаровані особини з цієї лінії змогли не лише зберегти свої корисні ознаки, але й покращити їх, що в решті решт привело до появи переможця еволюції.

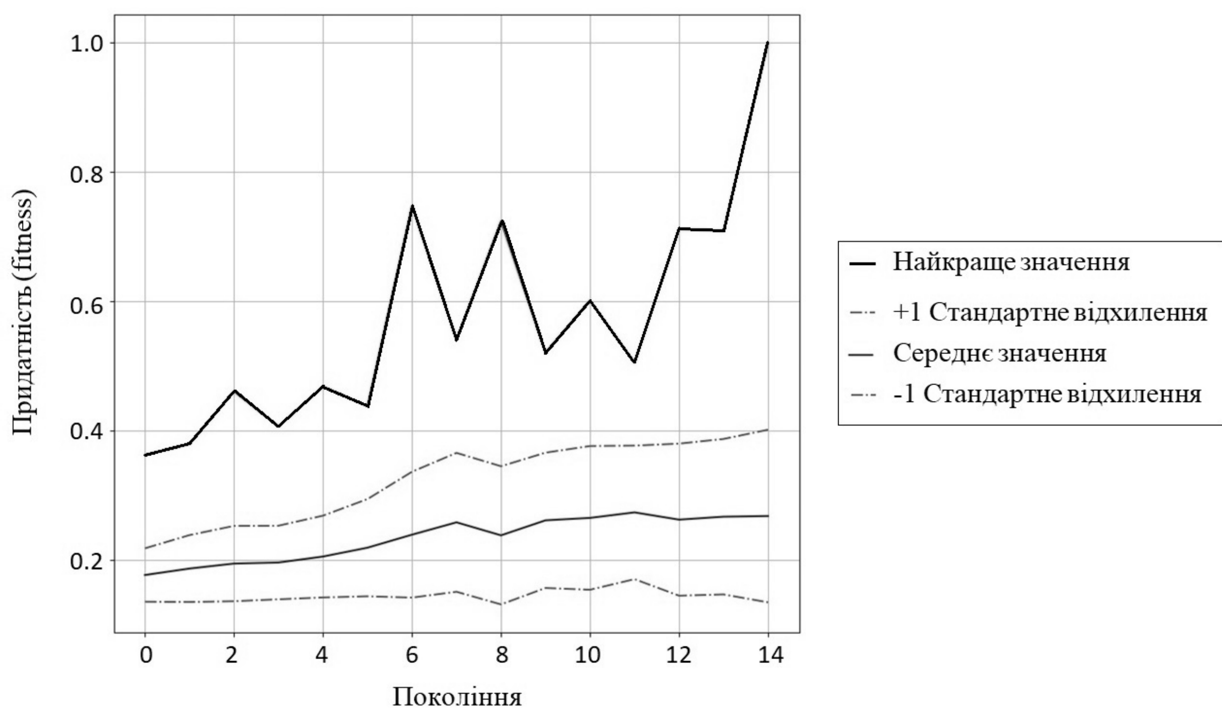


Рис. 6. Середнє і найкраще значення функції придатності в експерименті зі зворотним маятником

Висновки

Розглянуто розроблений алгебро-алгоритмічний інструментарій, що застосовується для проєктування та синтезу програм, які використовують нейроеволюційні алгоритми. В основу інструментарію покладено конструювання високорівневих специфікацій алгоритмів, поданих в системах алгоритмічних алгебр Глушкова, та генерацію відповідних програм на основі шаблонів реалізацій цільовою мовою програмування. Підхід проілюстровано на прикладі проєктування параметризованого алгоритму оцінювання для двійкового мультиплексора з подальшою реалізацією алгоритму мовою

C# для фреймворку SharpNEAT та на прикладі генерації програми для задачі балансування зі зворотним маятником, що використовує нейроеволюційний алгоритм бібліотеки NEAT-Python.

Література

1. Fogel D.B., Liu D., Keller J.M. Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation. Hoboken: Wiley-IEEE Press, 2016. 378 p.
2. Zgurovsky M.Z., Zaychenko Yu.P. Fundamentals of computational intelligence: system approach. Cham: Springer, 2016. 275 p.
3. Omelianenko I. Hands-on neuroevolution with Python. Birmingham: Packt, 2019. 368 p.
4. Lillicrap T.P., Santoro A., Marris C.J., Akerman C., Hinton G.E. Backpropagation and the brain. *Nature Reviews Neuroscience*. 2020. Vol. 21. P. 335–346.
5. Stanley K.O., Clune J., Lehman J., Miikkulainen R. Designing neural networks through neuroevolution. *Nature Machine Intelligence*. 2019. Vol. 1. P. 24–35. http://doi.org/10.1038/s42256-018-0006-z
6. Stanley K.O. Neuroevolution: a different kind of deep learning. 2017. URL: http://www.oreilly.com/radar/neuroevolution-a-different-kind-of-deep-learning
7. Андон П.І., Дорошенко А.Ю., Акуловський В.Г., Іваненко П.А., Яценко О.А. Формальні та адаптивні методи конструювання високопродуктивних паралельних програм. Київ: Наукова думка, 2023. 312 с.
8. Doroshenko A., Yatsenko O. Formal and adaptive methods for automation of parallel programs construction: emerging research and opportunities. Hershey: IGI Global, 2021. 279 p.
9. Andon P.I., Doroshenko A.Yu., Zhereb K.A., Yatsenko O.A. Algebra-algorithmic models and methods of parallel programming. Kyiv: Akademperiodyka, 2018. 192 p.
10. SharpNEAT. Evolution of Neural Networks. URL: http://sharpneat.sourceforge.io
11. NEAT-Python. URL: http://github.com/CodeReclaimers/neat-python
12. Дорошенко А.Ю., Ашур І.З. Розподілена реалізація методу нейроеволюції наростаючої топології. *Проблеми програмування*. 2021. № 3. С. 3–15.

References

1. Fogel, D. B., Liu, D., Keller, J. M. (2016). *Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation*. Hoboken: Wiley-IEEE Press.
2. Zgurovsky, M. Z., Zaychenko, Yu. P. (2016). *Fundamentals of computational intelligence: system approach*. Cham: Springer.
3. Omelianenko, I. (2019) *Hands-on neuroevolution with Python*. Birmingham: Packt
4. Lillicrap, T. P., Santoro, A., Marris, C. J., Akerman, C., Hinton, G. E. (2020). Backpropagation and the brain. *Nature Reviews Neuroscience*, 21, 335-346. doi: 10.1038/s41583-020-0277-3
5. Stanley, K. O., Clune, J., Lehman, J., Miikkulainen, R. (2019). Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1, 24-35. doi: 10.1038/s42256-018-0006-z
6. Stanley, K. O. (2017). *Neuroevolution: a different kind of deep learning* [Online]. Available: http://www.oreilly.com/radar/neuroevolution-a-different-kind-of-deep-learning
7. Andon, P. I., Doroshenko, A. Yu., Akylovsky, V. G., Ivanenko, P. A., Yatsenko, O. A. (2023). *Formal'ni ta adaptyvni metody konstruyuvannya vysokoproduktyvnykh paralel'nykh prohram*. Kyiv: Naukova dumka.
8. Doroshenko, A., Yatsenko, O. (2021). *Formal and adaptive methods for automation of parallel programs construction: emerging research and opportunities*. Hershey: IGI Global.
9. Andon, P. I., Doroshenko, A. Yu., Zhereb, K. A., Yatsenko, O. A. (2018). *Algebra-algorithmic models and methods of parallel programming*. Kyiv: Akademperiodyka.
10. SharpNEAT. *Evolution of neural networks* [Online]. Available: http://sharpneat.sourceforge.io
11. NEAT-Python [Online]. Available: http://github.com/CodeReclaimers/neat-python
12. Doroshenko, A. Yu., Achour, I. Z. (2021). *Rozpodilena realizatsiya metodu neyroevolyutsiyi narostayuchoyi topolohiyi*. Problemy prohramuvannya, (3), 3-15.

The article has been sent to the editors 18.07.23.
After processing 08.08.23.
Submitted for printing 10.08.23.